Time-Lock Puzzles from Lattices

Shweta Agrawal[1], Giulio Malavolta[2], and Tianwei Zhang[3,4]

[1]IIT Madras
[2]Bocconi University
[3]Max Planck Institute for Security and Privacy
[4]Ruhr University Bochum

Abstract

Time-lock puzzles (TLP) are a cryptographic tool that allow one to encrypt a message into the future, for a predetermined amount of time T. At present, we have only two constructions with provable security: One based on the repeated squaring assumption and the other based on obfuscation. Basing TLP on *any* other assumption is a long-standing question, further motivated by the fact that known constructions are broken by quantum algorithms.

In this work, we propose a new approach to construct time-lock puzzles based on lattices, and therefore with plausible post-quantum security. We obtain the following main results:

- In the preprocessing model, where a one-time public-coin preprocessing is allowed, we obtain a time-lock puzzle with encryption time $\log(T)$.
- In the plain model, where the encrypter does all the computation, we obtain a time-lock puzzle with encryption time $\sqrt{T}$.

Both constructions assume the existence of any sequential function f, and the hardness of the circular small-secret learning with errors (LWE) problem.

At the heart of our results is a new construction of succinct randomized encodings (SRE) for T-folded repeated circuits, where the complexity of the encoding is $\sqrt{T}$. This is the first construction of SRE where the overall complexity of the encoding algorithm is sublinear in the runtime T, and which is not based on obfuscation. As a direct corollary, we obtain a non-interactive RAM delegation scheme with sublinear complexity (in the number of steps T).

1 Introduction

A Time-Lock Puzzle (TLP) [RSW96] allows one to encrypt a message into the future, for a predetermined amount of time T. Cryptographically, a TLP must satisfy two properties: The time needed to *generate* a puzzle must be sublinear, ideally logarithmic, in T (efficiency) and the time needed to *solve* a puzzle must be greater than T (sequentiality). Importantly, sequentiality must hold even against massively parallel algorithms, which should have no advantage compared to a regular sequential solver. In other words, solving a puzzle must be a strictly sequential operation.

Recently, TLPs have been the subject of intense study and a large body of work has discovered a number of different applications, such as sealed-bid auctions, e-voting systems [MT19], fair contract signing [BN00], non-malleable commitments [LPS17], cryptocurrency payment systems [TMSS22], atomic swaps [TMMS22], distributed consensus algorithms [WXDS20], and byzantine

consensus protocols [SLM+23], to mention a few. Very recently, TLPs have been used in real-world cryptographic protocols, specifically in sealed-bid auctions [TAF+23] and voting [GSZB23] on blockchains.

Given the wide spectrum of applications and the central role played by TLPs in modern cryptography, it is important to ask what are the computational assumptions needed to construct time-lock puzzles. At present, there are two main recipes[1] to build time-lock puzzles: Either using the hardness of repeated squaring on groups of unknown order [RSW96], or using indistinguishability obfuscation [BGJ+16]. To make things worse, when considering TLPs with security against *quantum algorithms*, all schemes derived from the first approach are plainly insecure, whereas the second approach is left with a handful of candidate constructions of obfuscation [Agr19, WW21, GP21, BDGM23], whose security is based on assumptions that are not yet well-understood.

The objective of this work is to make progress on this problem, and place TLPs with post-quantum security on solid foundations.

1.1 Our Results

In this work, we propose the first constructions of TLPs based on lattices, with provable security against well-defined and falsifiable lattice assumptions. More in details, we obtain two main results:

- (Preprocessing Model) In the preprocessing model, we allow for a one-time, public-coin computation whose runtime can depend on T. This computation produces some small public parameters pp (of size independent of T) that anyone can use to encrypt a message for time T. In this model, we obtain a TLP scheme where the encryption time is logarithmic in T and the solving time is linear in T, ignoring multiplicative factors in the security parameter.
- (Plain Model) In the plain model, there is no preprocessing, and the encrypter does all the computation. In this model we obtain a TLP where the encryption time and the puzzle size are bounded by $\sqrt{T}$, again ignoring factors dependent on the security parameter.

Both of our constructions require the following cryptographic ingredients:

1. Any sequential function f of the form $f(x) = C(\dots C(x)\dots)$, for some fixed circuit C.
2. The hardness of the circular small-secret LWE problem [HLL23].

The circular small-secret LWE problem is a variant, recently introduced in the context of attribute-based encryption [HLL23], of the well-known circular LWE problem, which is required to bootstrap [Gen09] lattice-based Fully Homomorphic Encryption (FHE). Thus, our approach yields the *first* sublinear TLPs based on standard assumptions that can be conjectured post-quantum secure.

At a technical level, we obtain our constructions via a two-step process: (i) We first propose a new compiler, inspired by the work of [BGJ+16], where we show how to combine any sequential function with a succinct randomized encoding (SRE) for repeated circuit computations to obtain a time-lock puzzle. Such a compiler relies on a new *depth-preserving* version of the celebrated Goldreich-Levin theorem [GL89]. Next, (ii) we show how to construct SREs for repeated circuits from the above-mentioned assumptions. While the preprocessing SRE follows almost immediately

[1] We defer a more thorough treatment of existing approaches to construct time-lock puzzles to Section 2, and we keep this discussion deliberately informal.

as a corollary of prior works, the SRE without preprocessing and with $\sqrt{T}$ encoding complexity is the technically most involved part of our work, and we view it as our main technical contribution. Such scheme is heavily inspired by techniques originally developed in the context of code obfuscation [BDGM23], but in a regime where we can prove security against the circular small-secret LWE problem and the LWE problem. This is the *first SRE construction* with sublinear encoding time that does not rely on iO. We believe our techniques are likely to have other applications. As an example, we show that our SRE yields the first non-interactive garbled RAM scheme with sublinear encoding complexity (in the number of steps T) without using iO.

Along the way, we define and construct the notion of *range puncturable* pseudorandom function (PRF), which extends the celebrated puncturable PRF [SW14] to puncture an entire (possibly exponentially large) range of inputs. This primitive may be of independent interest.

1.2 Technical Overview

In the following we give a high-level overview of the main technical ideas introduced in this work, while keeping the overall discussion somewhat informal. For more precise statements, we refer the reader to the technical sections of the paper. As alluded to earlier, our TLP constructions follow a two-step process, where we first show how to compile a sequential function f and an SRE for repeated circuits into a TLP, and then we show how to instantiate the cryptographic building blocks. We follow a similar outline in this overview, starting from the former.

Time-Lock Puzzles from SRE for Repeated Circuits. Let us first recall that an SRE for repeated circuits takes as input a circuit C, a repetition parameter T, and an input x and returns an encoded tuple $(\tilde{C}, \tilde{x})$. Anyone can then evaluate the encoded tuple to obtain

$$\mathsf{Eval}(\tilde{C}, \tilde{x}) = \underbrace{C(\dots C}_{T-\text{times}}(x)\dots) = C_T(x)$$

but otherwise no further information is learned about x[2]. The point here is that the complexity of the *encoding* procedure should be less than that of the evaluation process. In our case, we require that the complexity of the encoding algorithm should be *sublinear* in T.

Assuming the existence of a T-folded sequential function $f : \mathcal{X} \to \mathcal{X}$, we now show how to encrypt a *random* message m for time T. We define the circuit $C(b, x, m, z, i)$ as follows:

- If $i = T + 1$ and $b = 0$ return m.
- Else if $i = T + 1$ and $b = 1$ return $x \oplus z$.
- Otherwise return $(b, f(x), m, z, i + 1)$.

We then simply define our TLP to be an SRE encoding for the T-fold repetition of C, on input $(0, x, m, 0, 1)$, where x is a uniformly sampled instance from the domain of f. It is easy to verify that the scheme is correct since

$$C_T(0, x, m, 0, 1) = m.$$

[2]Here and throughout this work, we always assume that SRE protects the privacy of the input, but the circuit C is always public. The stronger definition, where C is also hidden can be achieved using standard techniques (e.g., universal circuits).

To argue security, we first apply the following modifications to the encoded input

$$\begin{aligned}(\tilde{C}, \tilde{x}) = \mathsf{Encode}(C, (0, x, m, 0, 1)) &\equiv \mathsf{Encode}(C, (0, x, m \oplus f_T(x), 0, 1)) \\ &\approx \mathsf{Encode}(C, (0, x, m \oplus f_T(x), m, 1)) \\ &\approx \mathsf{Encode}(C, (1, x, m \oplus f_T(x), m, 1)) \\ &\approx \mathsf{Encode}(C, (1, x, 0, m, 1))\end{aligned}$$

where the first equality holds because m is uniformly sampled, and for the other steps the circuit is functionally equivalent and thus indistinguishability follows from the security of the SRE. Note that in the last hybrid, the puzzle is "effectively" encrypting $f_T(x) \oplus m$, by the definition of C. Now we can argue that any adversary that is able to output the message in time less than T will also compute $f_T(x)$, thus violating the sequentiality of f.

This argument crucially relies on m being sampled uniformly, and on the distinguisher being forced to output the entire message (rather than just distinguishing two distributions). Fortunately here we can apply the standard Goldreich-Levin transformation [GL89], which allows us to achieve the standard notion of security. A subtle point here is that we need to make sure that the Goldreich-Levin reduction does not introduce additional *depth* to the circuit of the reduction, thus voiding our above argument. For this reason, we prove a depth-preserving version of the standard Goldreich-Levin theorem, where the depth of the circuit is only marginally increased by the search-to-decision reduction. Finally, we also mention that, while we have described here a TLP in the plain model, a version with preprocessing follows from the same template, with minor syntactical adjustments.

Sublinear Randomized Encodings. Next, we turn to the question of constructing an SRE with the efficiency required to instantiate the above compiler. Relevantly for us, a recent work of Hsieh et al. [HLL23] shows (paraphrasing) how to construct an SRE with encoding time *logarithmic* in T from the circular small-secret LWE assumption. However, their construction requires a one-time public-coin preprocessing running in time T. Fortunately, such a preprocessing is reusable and therefore plugging it in the above compiler already yields a TLP with asymptotically optimal efficiency, in the preprocessing model. However, if we insist on a TLP in the plain model, the long preprocessing of [HLL23] brings us back to square one.

Our first observation is that, since the preprocessing is reusable, we can hope to *amortize the work*. We can then try to chop the execution of f_T in $\sqrt{T}$-many blocks, each of size $\sqrt{T}$, to reduce the complexity of the encoding. In more details, we can compute an SRE for the circuit $F_{\sqrt{T}}(x, i)$ defined as follows:

- If $i = T + 1$ return x.
- Else compute $y = f_{\sqrt{T}}(x)$ and output an *encoding* of $(y, i + \sqrt{T})$.

The idea is that we can run the preprocessing *once* for the circuit $F_{\sqrt{T}}$ and then compute an encoding of $(x, 1)$. The first evaluation would return a valid encoding of $(f_{\sqrt{T}}(x), \sqrt{T} + 1)$, which we can feed again into the evaluation algorithm to compute $(f_{2\sqrt{T}}(x), 2\sqrt{T} + 1)$, and so on, all the way until we compute $f_T(x)$.

While this almost seems like it would work, we have overlooked an important aspect: The size of an encoding in the [HLL23] scheme depends on the *output size* of the circuit.[3] Translating this

[3]This is to some extent inherent, since it is known that output compression implies obfuscation [AJ15, BV18].

to the above construction, this means that the size of an encoding grows *exponentially* with the number of repetitions! In other words, the scheme outlined above is not even running in polynomial time, beyond constant T. To fix this, we resort to ideas from constructions of obfuscation, and specifically to the notion of *split-FHE* [BDGM23]. Loosely speaking, a split-FHE scheme is a standard FHE, except that when evaluating a function

$$\mathsf{Enc}(m) \xrightarrow{g} \mathsf{Enc}(g(m))$$

one can compute a *small hint* $\mathsf{h}_{g,m}$ that allows one to decrypt the evaluated ciphertext. For the purpose of this work, it suffices to recall two properties about the hint:

1. (Succinctness) The size of the hint $\mathsf{h}_{g,m}$ is independent from the size of the output $g(m)$.
2. (Linearity) The hinted decryption operation is linear, that is:

$$\mathbf{A} \cdot \mathsf{h}_{g,m} - \mathsf{Enc}(g(m)) \approx g(m)$$

for some matrix $\mathbf{A}$, and where the approximate equality hides some noise terms.

Our idea is to *alternate* the computation of the SRE with the computation of the split-FHE. In other words, we modify the function $F_{\sqrt{T}}$ so that, instead of computing the output directly, it outputs the *hint* of the split-FHE computation. This way the evaluator can conduct the split-FHE computation in a *parallel thread*, and use the information output by the SRE to compute the new encoded input. Since the size of the hint is succinct (and in particular it does not depend on the output size), this allows us to avoid the circular dependency in the parameters of the above attempt.

This new idea places us on the right track, but there are still two problems to solve: First, the hint is computed using the secret key of the split-FHE, and second, the hint may potentially reveal some extra information about the plaintext m, jeopardizing the security of the scheme. To solve this first problem, in our actual construction, we will actually use $\sqrt{T}$ different split-FHE schemes, but with coins pseudorandomly sampled. The security proof will require a non-standard puncturing argument of a PRF where, instead of puncturing a single point, we will have to puncture out an entire *range of points*. We defer more details about this to a later point of this overview. To solve the second problem, we use a technique from [BDGM23], where instead of releasing $\mathsf{h}_{g,m}$, we instruct our SRE to compute a masked term $\mathsf{h}_{g,m} + \mathsf{rand}$ instead. Using the linearity of the hinted decryption, we see that:

$$\begin{aligned}\mathbf{A} \cdot (\mathsf{h}_{g,m} + \mathsf{rand}) - \mathsf{Enc}(g(m)) &= \mathbf{A} \cdot \mathsf{h}_{g,m} + \mathbf{A} \cdot \mathsf{rand} - \mathsf{Enc}(g(m)) \\ &\approx \mathbf{A} \cdot \mathsf{rand} - g(m).\end{aligned}$$

Note that the terms $\mathbf{A} \cdot \mathsf{rand}$ is independent from any intermediate step of the computation, which means that we can include it as part of the puzzle. This way, the evaluator can subtract the term, and proceed with the rest of the evaluation. To be more precise, we include with the puzzle a *rounded* version (component-wise) of such vector, to smudge the noise of the hinted decryption, which may contain harmful information. A standard argument shows that the above equality still holds with high probability.

This outline glosses over many subtleties of the construction, and the final scheme requires a few more ideas in order to avoid circular parameter dependencies and to make the proof go through. Here we will simply say that security proof proceeds by iteratively "programming" an encoded input at each step of the computation, after "erasing" the information that can potentially harm security in the previous steps. For more details, we refer the reader to Section 6.

Range Puncturable PRF. As discussed before, our construction requires a PRF where we can "puncture" keys to remove from the evaluation domain an entire range of points. Although technically this is syntactically allowed by the standard puncturable PRF syntax [SW14], the issue is that the size of the punctured keys would grow with the number of punctured points. For our application, it is crucial that the size of the punctured key remains *constant*. In this work, we show how to construct a *range puncturable PRF*, by modifying the celebrated [GGM86] construction. Recall that [GGM86] defines the output of the PRF recursively as

$$\mathsf{PRF}(s, x||b) \coloneqq G_b(\mathsf{PRF}(s, x)),\ b \in \{0,1\},$$

where for a single bit

$$\mathsf{PRF}(s, b) \coloneqq G_b(s),\ b \in \{0,1\}$$

and G_b is a length-doubly pseudorandom generator (PRG). In order to puncture at range $[0, i]$, the range punctured key is defined to be

$$K\{i+1\} = \{\mathsf{PRF}(s, i_0 i_1 \dots i_{k-1} 1) | i_k = 0,\ k \in [0, n-1],\ i = i_0 i_1 \dots i_{n-1}\}$$

Note that $K\{i+1\}$ is sufficient to compute the PRF at any point outside the range $[0, i]$, since we can expand the tree below the neighboring nodes by applying the PRG to the values of the nodes. Furthermore, we can compute $K\{i+2\}$ from $K\{i+1\}$ without increasing the size of the key, by descending one level and erasing the parent node, when appropriate. This allows us to run the iterative puncturing argument in our main construction.

New Application: Sublinear Garbled RAM. We also mention that our new SRE can be seen as a garbled RAM scheme with sublinear complexity (in the number of steps T). Recall that a garbled RAM protocol consists of a client holding a RAM program P^* and an input x and computes a short encoding $\mathrm{Enc}(P^*, x)$, that allows the evaluator to recover $P^*(x)$ and nothing more. Sublinearity requires that the runtime of the client should be sublinear in the runtime of P^* on x.

The construction follows by simply observing that the computation of a RAM program can be parsed as the T-folded execution of a circuit P with the syntax

$$P(D, \mathrm{state}, i_{\mathrm{read}}, i_{\mathrm{write}}, b_{\mathrm{read}}, b_{\mathrm{write}}) = (D', \mathrm{state}', i'_{\mathrm{read}}, i'_{\mathrm{write}}, b'_{\mathrm{read}}, b'_{\mathrm{write}})$$

which takes as input a memory database D, some state, indices for the read/write operations, and bits to read/write. To delegate a RAM program of this form, we can simply call our SRE construction on input $(0^{|D|}, x, 0, 0, 0, 0)$. As for the efficiency, the running time of the client is $\mathsf{poly}(\lambda, |x|, |D|) \cdot \sqrt{T}$ while the runtime of the server is $\mathsf{poly}(\lambda, |x|, |D|) \cdot T$. Prior to our work, all protocols for non-interactive sublinear garbled RAM without preprocessing required obfuscation [GHRW14, BGL$^+$15, CHJV15, CH16, CCC$^+$15].

1.3 Organization

This manuscript is organized as follows: In Section 2 we discuss known approaches to construct time-lock puzzles in more details. Section 3 outlines the necessary cryptographic preliminaries. In Section 4, we present our compiler that turns an SRE and a sequential function into a TLP. In Section 5, we define and construct range puncturable PRFs. Section 6 details the construction of our main SRE scheme with sublinear encoding complexity, while Section 7 demonstrates the application of SRE to RAM Delegation.

2 Related Work

To better understand the context of our work, we discuss in details what are the known construction strategies to build TLPs, and why they do not suffice for our goal. At a high-level we can group existing constructions into three categories, which we describe below.

(1) The Group-Based Approach. The pioneering work of Rivest, Shamir, and Wagner [RSW96] builds the first TLPs in RSA groups, by conjecturing that repeated squaring is a sequential operation in groups of unknown order and using the group structure of RSA to "encrypt a message to the future". Ever since, variants of this construction have been proposed, adding properties such as homomorphism [MT19, BDGM19], batch-solving [SLM+23, DGM23], non-malleability [KLX20, FKPS21], verifiability [TBM+20, AK21], and adopting other group structures, such as class groups of imaginary quadratic order [TCLM21] or isogenies on elliptic curves [BDF21]. One property that all of these constructions have in common is that they are known to be insecure against quantum attacks.

(2) The Succinct Randomized Encoding Approach. The work of Bitansky et al. [BGJ+16] introduces an alternative method for constructing TLPs, which relies on the existence of succinct randomized encodings [BGL+15] and non-parallelizable languages. At present, the only constructions of succinct randomized encodings that can be used in their compiler rely on indistinguishability obfuscation [BGL+15]. Although lattice-based constructions of obfuscation exist [Agr19, WW21, GP21, BDGM23], the underlying assumptions are not well-understood and it is considered an open problem to build lattice-based obfuscation based on a "standard" assumption.

Our work is closely related to this approach, and can be thought of as constructing the kind of randomized encodings that are sufficient to instantiate a *variant* of their compiler. In fact, one contribution of our work is to describe an alternative compiler to build time-lock puzzle (Section 4), based on weaker premises. Compared to their approach, our compiler has two main advantages: (i) It provides (provably) meaningful security also in the preprocessing settings, whereas [BGJ+16] only proves a weaker notion of security. More in details, in [BGJ+16] the message is shown be hidden for a given time starting from the publication of the *public parameters*, whereas we prove that the same holds starting from the publication of the *puzzle*. [BGJ+16] leaves as an open problem to prove the security of their compiler in the stronger, reusable, settings. (ii) Our compiler does not require the full power of succinct randomized encodings, but only randomized encodings for *repeated computations*, which is a weaker functionality. On the flip side, our compiler is less general since it requires an explicit sequential function f, as opposed to the mere existence of one.

(3) The Witness Encryption Approach. A folklore construction of TLP uses *extractable* witness encryption [GGSW13], succinct non-interactive arguments for deterministic computation [CJJ22], and the existence of a sequential function f. In brief, the encryptor computes a witness encryption for the statement that checks whether one knows a valid proof for the sequential evaluation of f. Anyone that successfully evaluates f, can produce a proof of this fact, which functions as the secret key for the witness encryption ciphertext. Crucially, checking the validity of a proof does not require one to recompute the function f, and therefore this scheme satisfies the desired efficiency requirements. It is also important that the witness encryption scheme should be

extractable, so that a reduction can efficiently derive a contradiction against the soundness of the succinct argument.

The works of Liu et al. [LJKW18] and of Döttling et al [DHMW23] can be seen as a special case of this general paradigm. Succinct arguments can be built from lattices [CJJ22, ACL+22, CLM23], and recent works have proposed candidate constructions of witness encryption [VWW22, Tsa22], based on *evasive* lattice assumptions but, at present, no provable construction of *extractable* witness encryption is known, and there is evidence against the existence of this primitive [GGHW17]. Thus, although this approach leads to at least a scheme with heuristic security, a proof from a standard assumption seems out of reach.

3 Preliminaries

Throughout this work, we write λ to denote the security parameter. A function $\mu : \mathbb{N} \to \mathbb{N}$ is said to be negligible, denoted $\mu(n) = \mathrm{negl}(n)$, if for every positive polynomial $p(\cdot)$ and all sufficiently large n it holds that $\mu(n) < 1/p(n)$. We say an algorithm is efficient if it runs in probabilistic polynomial time (PPT) in the length of its input. In this work, we model all algorithms as circuits, and we refer to their size as the number of gates, whereas their depth is the depth of the circuit. We denote by $[n]$ the set $\{1, \ldots, n\}$.

3.1 Depth-Preserving Goldreich-Levin

In the following we recall the celebrated Goldreich-Levin theorem [GL89] and we highlight a property of the extractor that is implicitly present in the original proof. Namely, that the circuit depth of the extractor is poly-logarithmic in the inverse of the success probability of the algorithm.

Theorem 3.1 ([GL89])**.** Let $\mathcal{A}_x$ be an algorithm such that

$$\Pr_{r \leftarrow \{0,1\}^\lambda} \left[\mathcal{A}_x(r) = \bigoplus_{i=1}^{n} x_i r_i \right] \geq \frac{1}{2} + \varepsilon.$$

Then there exists an extractor $x \leftarrow \mathcal{E}^{\mathcal{A}_x}$ whose success probability and runtime are a polynomial in $1/\varepsilon$. Furthermore, the depth of the extractor is bounded by that of $\mathcal{A}$ plus poly-logarithmic in $1/\varepsilon$.

Proof Sketch. We briefly recall how the Goldreich-Levin extractor works. On input ε, and for each $i = 1, \ldots, \lambda$, the extractor prepares a set of correlated queries for $\mathcal{A}_x$. The size of the set is at most $1/\varepsilon^{O(1)}$ and all queries are issued in parallel to $\mathcal{A}_x$. Then, the extractor sets the i-th bit of x to be the majority bit of the answers of $\mathcal{A}_x$. Since all queries are asked in parallel to $\mathcal{A}_x$ and the majority function can be computed in depth logarithmic in the input size, the statement follows. □

3.2 Lattices and Learning with Errors

We recall the definition of the decisional Learning with Errors problem, introduced by Regev [Reg09].

Definition 3.2 (Decisional Learning with Errors)**.** Let $n = n(\lambda)$ and $q = q(\lambda)$ be integer parameters and $\chi = \chi(\lambda)$ be a distribution over $\mathbb{Z}$. The $\mathrm{LWE}_{n,m,q,\chi}$ problem is to distinguish between the distributions

$$\{\mathbf{A}, \mathbf{As} + \mathbf{e} \pmod q\} \quad \text{and} \quad \{\mathbf{A}, \mathbf{u}\}$$

where $\mathbf{A} \leftarrow \mathbb{Z}_q^{n \times m}$, $\mathbf{s} \leftarrow \mathbb{Z}_q^n$, $\mathbf{e} \leftarrow \chi^m$ and $\mathbf{u} \leftarrow \mathbb{Z}_q^m$. The Learning with Errors (LWE) assumption is that no polynomial time procedure can solve the $\mathrm{LWE}_{n,m,q,\chi}$ problem with more than a negligible advantage in λ.

We rely on LWE security with the following range of parameters. Let $\chi = \chi(\lambda)$ be a distribution supported in $[-B, B]$ such that $q/B \approx 2^{\omega(\log \lambda)}$ is super polynomial. The works of [Reg09, Pei09] showed that the LWE assumption with the above parameters follows from the worst-case quantum hardness SIVP and classical hardness of GapSVP with sub-exponential approximation factors.

3.3 Time-Lock Puzzles

We recall the definition of standard time-lock puzzles [BGJ+16]. For conceptual simplicity we consider only schemes with binary solutions.

Definition 3.3 (Time-Lock Puzzles)**.** A time-lock puzzle is a tuple of two algorithms $(\mathsf{PGen}, \mathsf{PSolve})$ defined as follows.

- $Z \leftarrow \mathsf{PGen}(T, s)$ a probabilistic algorithm that takes as input a hardness-parameter T and a solution $s \in \{0, 1\}$, and outputs a puzzle Z.
- $s \leftarrow \mathsf{PSolve}(Z)$ a deterministic algorithm that takes as input a puzzle Z and outputs a solution s.

Definition 3.4 (Correctness)**.** For all $\lambda \in \mathbb{N}$, for all polynomials T in λ, and for all $s \in \{0, 1\}$, it holds that

$$s = \mathsf{PSolve}(\mathsf{PGen}(T, s)).$$

Definition 3.5 (Security)**.** A scheme $(\mathsf{PGen}, \mathsf{PSolve})$ is secure if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of depth $o(T)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that

$$\Pr\left[b \leftarrow \mathcal{A}(Z) : Z \leftarrow \mathsf{PGen}(T(\lambda), b)\right] \leq \frac{1}{2} + \mu(\lambda).$$

In terms of efficiency, we require that the time to compute the puzzle should be sublinear, ideally polylogarithmic, in T.

Time-Lock Puzzle with Setup. We also consider a relaxed variant of time-lock puzzles, where the syntax is augmented with an additional setup algorithm PSetup that computes public parameters that are made available to all algorithms. Specifically we define the additional algorithm as:

- $\mathsf{pp} \leftarrow \mathsf{PSetup}(1^\lambda, T)$ a probabilistic algorithm that takes as input a security parameter 1^λ and a time hardness parameter T, and outputs public parameters pp.

We then modify the security notion accordingly.

Definition 3.6 (Reusable Security)**.** A scheme $(\mathsf{PSetup}, \mathsf{PGen}, \mathsf{PSolve})$ is reusable secure if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $(\mathcal{A}_1, \mathcal{A}_2) = \{(\mathcal{A}_1, \mathcal{A}_2)_\lambda\}_{\lambda \in \mathbb{N}}$ where the depth of $\mathcal{A}_2$ is bounded from above by $o(T)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that

$$\Pr\left[b \leftarrow \mathcal{A}_2(Z, \mathbf{z}) : \begin{array}{l} \mathsf{pp} \leftarrow \mathsf{PSetup}(1^\lambda, T(\lambda)) \\ \mathbf{z} \leftarrow \mathcal{A}_1(\mathsf{pp}) \\ b \leftarrow \{0,1\} \\ Z \leftarrow \mathsf{PGen}(\mathsf{pp}, b) \end{array}\right] \leq \frac{1}{2} + \mu(\lambda).$$

3.4 Reusable Garbled Circuits

We recall the definition of reusable garbled circuits [GKP+13].

Definition 3.7 (Reusable Garbled Circuits)**.** A reusable garbling scheme $(\mathsf{rGC.Garble}, \mathsf{rGC.Enc}, \mathsf{rGC.Eval})$ for circuits is defined as the following tuple of algorithms.

$(\tilde{C}, \mathsf{pk}) \leftarrow \mathsf{rGC.Garble}\left(1^\lambda, C\right)$**:** Given the security parameter 1^λ and a circuit C, the garbling algorithm returns a garbled circuit $\tilde{C}$ and an encoding key pk.

$\tilde{x} \leftarrow \mathsf{rGC.Enc}(\mathsf{pk}, x)$**:** Given the encoding key pk and an input x, the garbling algorithm returns an encoded input $\tilde{x}$.

$y \leftarrow \mathsf{rGC.Eval}\left(\tilde{C}, C, \tilde{x}\right)$**:** Given a garbled circuit $\tilde{C}$, and garbled input $\tilde{x}$, the $\mathsf{rGC.Eval}$ returns a string y.

In the following we define correctness for the reusable garbled circuit. For simplicity, we just define perfect correctness, but the definition can be easily adapted to weaker notions of statistical and computational correctness. This is relevant since the reusable garbled circuit presented in [HLL23] achieves only computational correctness. These differences will be largely irrelevant for us, since our results will hold assuming any of these notions of correctness.

Definition 3.8 (Correctness)**.** A garbling scheme $(\mathsf{rGC.Garble}, \mathsf{rGC.Eval})$ is correct if for all C and for all inputs x

$$C(x) = \mathsf{rGC.Eval}\left(\tilde{C}, C, \mathsf{rGC.Enc}(pk, x)\right),$$

where $(\tilde{C}, \mathsf{pk}) \leftarrow \mathsf{rGC.Garble}\left(1^\lambda, C\right)$.

Finally, we define security for a reusable garbled circuit.

Definition 3.9 (Security)**.** There exists a negligible function $\mu(\cdot)$ and an efficient simulator Sim such that for any polynomial time adversary $\mathcal{A}$, any circuit C and input x,

$$\left|\begin{array}{l} \Pr\left[\mathcal{A}\left(\tilde{C}, \mathsf{pk}, \mathsf{rGC.Enc}(\mathsf{pk}, x)\right) = 1\right] \\ -\Pr\left[\mathcal{A}\left(\tilde{C}, \mathsf{pk}, \mathsf{Sim}\left(1^\lambda, C, \mathsf{pk}, C(x)\right)\right) = 1\right] \end{array}\right| \leq \mu(\lambda),$$

where $(\tilde{C}, \mathsf{pk}) \leftarrow \mathsf{rGC.Garble}\left(1^\lambda, C\right)$.

We also remark that, by making the $\mathsf{rGC.Enc}$ algorithm part of the $\mathsf{rGC.Garble}$ algorithm, we recover the standard definition of (non-reusable) randomized encodings [Yao82, Yao86] where the syntax and the properties are changed accordingly. In this work we will use both notions (for instance in Section 6 we construct a non-reusable randomized encoding), but we omit a formal definition for the latter.

Depth-Independent Reusable Garbling. We explicitly recall a recent work [HLL23] that builds reusable garbled circuits where the complexity of the encoding algorithm is *entirely* independent from the complexity of the circuit. The construction is proven secure assuming the hardness of the *circular small-secret LWE* problem, which is a variant of the standard circular LWE assumption used to construct fully-homomorphic encryption. We refer the reader to [HLL23] for more details on the assumption.

Theorem 3.10 ([HLL23])**.** Assuming circular small-secret LWE assumption, there exists a reusable garbling scheme with

$$|\tilde{C}| = \mathsf{poly}(\lambda) \cdot |y| \quad \text{and} \quad |\mathsf{pk}| = \mathsf{poly}(\lambda) \cdot |x| \cdot |y| \quad \text{and} \quad |\tilde{x}| = \mathsf{poly}(\lambda) \cdot |x| \cdot |y|$$

and furthermore

$$|\mathsf{rGC.Garble}| = \mathsf{poly}(\lambda) \cdot |C| \quad \text{and} \quad |\mathsf{rGC.Enc}| = \mathsf{poly}(\lambda) \cdot |x| \cdot |y| \quad \text{and} \quad |\mathsf{rGC.Eval}| = \mathsf{poly}(\lambda) \cdot |C|$$

where $|C|$ is the number of gates of C (including input and output gates).

In this work we are actually going to use a slightly different version of this result, applying a transformation from [QWW18] (Section 4.2.2 in [QWW18]). The transformation uses the concept of laconic function evaluation (LFE), and we sketch it here: Instead of outputting directly pk, the $\mathsf{rGC.Garble}$ algorithm returns digest, which is the output of the $\mathsf{LFE.Compress}$ algorithm, on input the function $\mathsf{rGC.Enc}(\mathsf{pk}, \cdot)$. The new $\mathsf{rGC.Enc}$ algorithm then takes as input digest and x and computes the ciphertext ct_x via the $\mathsf{LFE.Enc}$ algorithm. The new $\mathsf{rGC.Eval}$ algorithm first runs $\mathsf{LFE.Dec}(\mathsf{rGC.Enc}(\mathsf{pk}, \cdot), \mathsf{ct}_x)$ to get $\tilde{x}$ and then continue with the original $\mathsf{rGC.Eval}$ algorithm. Note that the asymptotic complexity of the algorithms is not changed by this transformation. Furthermore, since LFE can be constructed from LWE [QWW18], the underlying computational assumption is unchanged as well.

Given that the encoding algorithm of the compiled construction is identical to $\mathsf{LFE.Enc}$ algorithm, we can analyze the components of an input encoding by inspecting the construction of [QWW18] (Appendix E in [QWW18]). For an input x, an encoding contains the following input dependent components:

$$\left\{\Psi_i = \begin{bmatrix} \mathbf{B}_i \\ \mathbf{s}^\intercal \mathbf{B}_i + \mathbf{e}_i \end{bmatrix} + x_i \cdot \mathbf{G}\right\}_i \quad \text{and} \quad \left\{\mathbf{b}_j = \mathbf{s}^\intercal(\mathbf{A}_j - \psi_j \cdot \bar{\mathbf{G}}) + \mathbf{e}_j^\intercal\right\}_j$$

where the matrices $\mathbf{A}_j$ are part of the public key, whereas the matrices $\mathbf{B}_j$, the vector $\mathbf{s}$, and the error terms are taken from the randomness. Here ψ is the bit-decomposition of Ψ and $\bar{\mathbf{G}}$ is the gadget matrix without the last row. In addition, denoting by C_k the circuit that computes the k-th output bit, the encoding contains the following input-independent components:

$$\{\beta_k = \mathbf{s}^\intercal \mathbf{A}_{C_k} \mathbf{t}_k + \tilde{e}_k, \mathbf{t}_k\}_k$$

where the matrices $\mathbf{A}_{C_k}$ can be thought of as the garbled circuit. Denoting by rand the randomness used by the encoding algorithm, we can split the execution of the input encoding algorithm $\mathsf{rGC.Enc}$ into three subroutines:

- $\mathsf{rGC.PartEnc}(\mathsf{pk}, \mathsf{rand})$: The partial encoding algorithm computes the input-independent components β_k and $\mathbf{t}_k$, along with

$$\left\{\Psi_i = \begin{bmatrix} \mathbf{B}_i \\ \mathbf{s}^\intercal \mathbf{B}_i + \mathbf{e}_i \end{bmatrix} + \mathbf{P}_i \right\}_i \quad \text{and} \quad \left\{\mathbf{b}_j = \mathbf{s}^\intercal \mathbf{A}_j + \mathbf{p}_j + \mathbf{e}^\intercal\right\}_j$$

  where $\mathbf{P}_i$ and $\mathbf{p}_j$ are uniformly sampled from the appropriate domain. Denote by $\tilde{x}_{part}$ the partial encoding.

- $\mathsf{rGC.InputEnc}(x, \mathsf{rand})$: The input encoding algorithm computes

$$\{x_i \cdot \mathbf{G} - \mathbf{P}_i\}_i \quad \text{and} \quad \left\{-\mathbf{s}^\intercal \psi_j \cdot \bar{\mathbf{G}} - \mathbf{p}_j\right\}_j$$

  and returns $\tilde{x}_{input}$ as the input encoding.

- $\mathsf{rGC.RecEnc}(\tilde{x}_{part}, \tilde{x}_{input})$: Compute the encoded input $\tilde{x}$ by adding the corresponding components of $\tilde{x}_{part}$ and $\tilde{x}_{input}$.

It is easy to see that

$$\mathsf{rGC.Enc}(\mathsf{pk}, x; \mathsf{rand}) = \mathsf{rGC.RecEnc}(\mathsf{rGC.PartEnc}(\mathsf{pk}, \mathsf{rand}), \mathsf{rGC.InputEnc}(x, \mathsf{rand})).$$

Furthermore, there exists a simulator $\mathsf{rGC.SimEnc}$ such that for all input x it holds that

$$\mathsf{rGC.SimEnc}(\mathsf{rGC.Enc}(\mathsf{pk}, x; \mathsf{rand})) \equiv (\mathsf{rGC.PartEnc}(\mathsf{pk}, \mathsf{rand}), \mathsf{rGC.InputEnc}(x, \mathsf{rand}))\,.$$

On input an encoding $\tilde{x} = (\Psi_i, \mathbf{b}_j, \mathbf{t}_k, \beta_k)$, the simulator $\mathsf{rGC.SimEnc}$ simulates the input-dependent components of $\tilde{x}_{part}$ with random matrices $\mathbf{R}_i$ and random vectors $\mathbf{r}_j$, and computes the corresponding elements of $\tilde{x}_{input}$ as

$$\{\Psi_i - \mathbf{R}_i\}_i \quad \text{and} \quad \{\mathbf{b}_j - \mathbf{r}_j\}_j\,.$$

This distribution is identical to that induced by the above algorithms.

3.5 Sequential Functions

We assume the existence of a sequential function f, i.e., a function

$$f : \{0,1\}^\lambda \to \{0,1\}^\lambda$$

such that, for a uniformly sampled input $x \in \{0,1\}^\lambda$ computing

$$y = \underbrace{f(\ldots f(f}_{T \text{ times}}(x))\ldots)$$

takes time $\Omega(T)$, for any efficient adversary. For notational convenience, we use the shorthand f_T to denote the T-fold repetition of f. We formalize this notion below.

Definition 3.11 (Security)**.** A function f is sequential if there exists a polynomial $\tilde{T}(\cdot)$ such that for all polynomials $T(\cdot) \geq \tilde{T}(\cdot)$ and every polynomial-size adversary $\mathcal{A} = \{\mathcal{A}_\lambda\}_{\lambda \in \mathbb{N}}$ of depth $o(T)$, there exists a negligible function $\mu(\cdot)$, such that for all $\lambda \in \mathbb{N}$ it holds that

$$\Pr\left[f_T(x) \leftarrow \mathcal{A}(x) : x \leftarrow \{0,1\}^\lambda\right] \leq \mu(\lambda).$$

Candidate sequential functions of this form have been proposed in the literature, and we recall a here few candidate functions.

- It is well-known that a random oracle is sequential (unconditionally), and therefore it is a widely accepted conjecture that setting f to be some cryptographic hash function (e.g., SHA-256) yields a heuristic sequential function.
- It was shown in [JMRR21] that the repeated evaluation of a dummy circuit in a fully homomorphic encryption scheme is sequential, if sequential function exists. In other words, assuming the hardness of the circular LWE assumption (needed to build fully-homomorphic encryption) there exists an explicit *universal* sequential function.
- Finally, a recent work [LM23] proposes the function

  $$f_{\mathbf{A}}(\mathbf{x}) = \mathbf{G}^{-1}(\mathbf{A}\mathbf{x})$$

  where $\mathbf{G}^{-1}$ denotes the bit decomposition operator, as candidate lattice-based sequential function.

3.6 Decrypt-and-Multiply Homomorphic Encryption

We recall the definition of Decrypt-and-Multiply homomorphic encryption scheme [BDGM19] in the following. Such scheme satisfies a fine-grained correctness property, which requires that the decryption consists of the application of a linear function in the secret key, followed by a rounding. Furthermore, we require that such a procedure allows us to specify an arbitrary constant ω that is multiplied to the resulting plaintext. We stress that all major FHE constructions satisfy (or can be adapted to) such a constraint, e.g., [GSW13, ASP13, BV14].

Definition 3.12 (Decrypt-and-Multiply Homomorphic Encryption)**.** A Decrypt-and-Multiply homomorphic encryption scheme consists of the following efficient algorithms.

- $\mathsf{KeyGen}(1^\lambda)$: On input the security parameter 1^λ, the key generation algorithm returns a key pair $(\mathsf{sk}, \mathsf{pk})$.
- $\mathsf{Enc}(\mathsf{pk}, m)$: On input a public key pk and a message m, the encryption algorithm returns a ciphertext c.
- $\mathsf{Eval}(\mathsf{pk}, C, (c_1, \ldots, c_\ell))$: On input the public key pk, an l-inputs circuit C, and a vector of ciphertexts $(c_1, \ldots, c_\ell)$, the evaluation algorithm returns an evaluated ciphertext c.
- $\mathsf{Dec}(\mathsf{sk}, c)$: On input the secret key sk and a ciphertext c, the decryption algorithm returns a message m.

- $\mathsf{Dec\&Mult}(\mathsf{sk}, c, \omega)$: On input the secret key sk, the Decrypt-and-Multiply consists of the application of a linear function in the secret key, followed by some publicly computable function, and it outputs a message $\tilde{m}$.

Definition 3.13 (Correctness)**.** A Decrypt-and-Multiply homomorphic encryption scheme (KeyGen, Enc, Eval, Dec, $\mathsf{Dec\&Mult}$) is correct if for all $\lambda \in \mathbb{N}$, all l-inputs circuits C, all inputs $(m_1, \ldots, m_\ell)$, all $(\mathsf{sk}, \mathsf{pk})$ in the support of $\mathsf{KeyGen}(1^\lambda)$, and all c_i in the support of $\mathsf{Enc}(\mathsf{pk}, m_i)$ it holds that

$$Pr[\mathsf{Dec}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, C, (c_1, \ldots, c_\ell))) = C(m_1, \ldots, m_\ell)] = 1.$$

and

$$\mathsf{Dec\&Mult}(\mathsf{sk}, \mathsf{Eval}(\mathsf{pk}, C, (c_1, \ldots, c_\ell)), \omega) = \omega \cdot C(m_1, \ldots, m_\ell) + e \mod q$$

where $\mathsf{Dec\&Mult}$ is a linear function in sk and $|e| \leq B$ with all but negligible probability.

We require a scheme to be compact in the sense that the size of the ciphertext should not grow with the size of the evaluated circuit.

Definition 3.14 (Compactness)**.** A Decrypt-and-Multiply homomorphic encryption scheme is compact if there exists a polynomial $\mathsf{poly}(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all ℓ-inputs circuits C in the supported family, all inputs $(m_1, \ldots, m_\ell)$, all $(\mathsf{sk}, \mathsf{pk})$ in the support of $\mathsf{KeyGen}(1^\lambda)$, and all c_i in the support of $\mathsf{Enc}(\mathsf{pk}, m_i)$ it holds that

$$|\mathsf{Eval}(\mathsf{pk}, C, (c_1, \ldots, c_\ell))| = \mathsf{poly}(\lambda) \cdot |C(m_1, \ldots, m_\ell)|.$$

Definition 3.15 (Semantic Security)**.** A Decrypt-and-Multiply homomorphic encryption scheme is semantically secure if for all polynomial-size distinguishers $\mathcal{A}$ there exists a negligible function $\mu(\cdot)$ such that for all $\lambda \in \mathbb{N}$, all pairs of message (m_0, m_1), it holds that

$$|\Pr[1 = \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_0))] - \Pr[1 = \mathcal{A}(\mathsf{pk}, \mathsf{Enc}(\mathsf{pk}, m_1))]| = \mu(\lambda)$$

where $(\mathsf{sk}, \mathsf{pk}) \leftarrow \mathsf{KeyGen}(1^\lambda)$.

Most known constructions of fully-homomorphic encryption are captured in this framework, for instance [GSW13].

4 Time-Lock Puzzle Construction

In the following, we present our time-lock puzzle construction. Our construction will use the following ingredients:

- A function f, whose T-folded repetition $f_T(x) = f(\ldots f(x) \ldots)$ is inherently sequential.
- A reusable garbled circuits ($\mathsf{rGC.Garble}, \mathsf{rGC.Enc}, \mathsf{rGC.Eval}$).

We describe the construction in the reusable setting, assuming a setup whose runtime can be proportional to T. However, exactly the same construction gives a time-lock puzzle without setup if we use the sublinear randomized (Section 6) instead of the reusable garbled circuits, and merge the setup with the puzzle generation algorithm.

- $\mathsf{PSetup}(1^\lambda, T)$: Compute $(\tilde{C}, \mathsf{pk}) \leftarrow \mathsf{rGC.Garble}\left(1^\lambda, C_T\right)$ where the circuit $C(b, x, m, z, i)$ is defined as follows.
  - If $i = T + 1$:
    - If $b = 0$: Return m.
    - If $b = 1$: Return $x \oplus z$.
  - Otherwise, return $(b, f(x), m, z, i + 1)$.

  And we denote by C_T the T-fold repetition of C. Set the public parameters to $\mathsf{pp} = (\tilde{C}, \mathsf{pk})$.
- $\mathsf{PGen}(\mathsf{pp}, s)$: Given $s \in \{0,1\}$, Sample a random $x \leftarrow \{0,1\}^\lambda$, a random $m \leftarrow \{0,1\}^\lambda$, and a random $r \leftarrow \{0,1\}^\lambda$. Compute

  $$\tilde{x} \leftarrow \mathsf{rGC.Enc}(\mathsf{pk}, (0, x, m, 0^\lambda, 1))$$

  and return $Z = (\tilde{x}, r, r \cdot m \oplus s)$, where $r \cdot m$ is the Goldreich-Levin inner-product predicate.
- $\mathsf{PSolve}(Z)$: Compute

  $$y \leftarrow \mathsf{rGC.Eval}\left(\tilde{C}, C_T, \tilde{x}\right)$$

  and use $y \cdot r$ to unmask the secret s.

Correctness follows by a straightforward invocation of the correctness of the underlying building blocks. In the following we show that solving the time-lock puzzle takes time indeed proportional to T.

Theorem 4.1. Let rGC be a reusable garbled circuit and let f be a sequential function. Then the construction as described above is a reusably-secure time-lock puzzle.

Proof. By Theorem 3.1, it suffices to show that there exists no adversary of depth sublinear in T that, on input $\tilde{x}$, can return $m \in \{0,1\}^\lambda$ with non-negligible probability. Recall that in the original experiment, we compute

$$\tilde{x} = \mathsf{rGC.Enc}(\mathsf{pk}, (0, x, m, 0^\lambda, 1)).$$

We are now going to change the distribution of $\tilde{x}$ via hybrid experiments, and we argue that each modification is computationally indistinguishable. Specifically, we observe that

$$\begin{aligned}
\tilde{x} = \mathsf{rGC.Enc}(\mathsf{pk}, (0, x, m, 0^\lambda, 1)) &\equiv \mathsf{rGC.Enc}(\mathsf{pk}, (0, x, \eta \oplus f_T(x), 0^\lambda, 1)) \\
&\approx \mathsf{rGC.Enc}(\mathsf{pk}, (0, x, \eta \oplus f_T(x), \eta, 1)) \\
&\approx \mathsf{rGC.Enc}(\mathsf{pk}, (1, x, \eta \oplus f_T(x), \eta, 1)) \\
&\approx \mathsf{rGC.Enc}(\mathsf{pk}, (1, x, 0^\lambda, \eta, 1))
\end{aligned}$$

where we have implicitly defined $\eta = m \oplus f_T(x)$. It can be easily checked that the circuit C with the above inputs always returns the same output, and therefore indistinguishability follows by the security of the (reusable) garbling scheme. We stress that computational indistinguishability follows for all PPT adversaries, i.e., not necessarily depth-bounded. At this point, we can show that the puzzle is sequential with a straightforward reduction against the sequentiality of f: The reduction samples a random $\eta \leftarrow \{0,1\}^\lambda$, computes $\tilde{x}$ as specified above, and activates the adversary, who

returns some m^*. The reduction returns $m^* \oplus \eta$. If the adversary returns the correct output of the circuit (which happens with at least inverse-polynomial probability), then the output of the reduction equals

$$C_T(1, x, 0^\lambda, \eta, 1) \oplus \eta = \eta \oplus f_T(x) \oplus \eta = f_T(x)$$

as desired. □

We also state a theorem for our construction without pre-processing, where we substitute the reusable garbled circuit with a sublinear randomized encoding (see Section 6 for definitions). Since the analysis is identical, up to syntactical modifications, we omit the proof.

Theorem 4.2. Let $(\mathsf{Enc}, \mathsf{Dec})$ be a sublinear randomized encoding and let f be a sequential function. Then the construction as described above (with PSetup integrated with PGen) is a secure time-lock puzzle.

5 Range Puncturable Pseudorandom Functions

A range puncturable psuedorandom function (PRF) is a PRF augmented with a procedure that allows to "puncture" a PRF key at a range $[0, i]$, in such a way that the adversary with the punctured key can evaluate the PRF at all points except the points in $[0, i]$. Moreover, even given the punctured key, an adversary cannot distinguish between a uniformly random value and the evaluation of the PRF at a point in $[0, i]$ with respect to the original unpunctured key. We provide a formal definition in the following.

Definition 5.1 (Range Puncturable PRF)**.** A range Puncturable psudorandom function $\mathsf{PRF} : \mathcal{K} \times \{0,1\}^{n(\lambda)} \to \{0,1\}^{m(\lambda)}$ consists of three polynomial time algorithms:

- $\mathsf{KeyGen}(1^\lambda)$: on input 1^λ, output a secret key $K \leftarrow \mathcal{K}$.
- $\mathsf{RangePuncture}(K, i)$: On input secret key K and index $i \in \{0,1\}^{n(\lambda)}$, output a punctured key $K\{i+1\}$.
- $\mathsf{IterPuncture}(K\{i\}, i)$: On input punctured key $K\{i\}$ at range $[0, i-1]$ and index $i \in \{0,1\}^{n(\lambda)}$, output a punctured key $K\{i+1\}$ at range $[0, i]$.

We require the range puncturable PRF to satisfy the following conditions:

- (Functionality Preserved Under Puncturing) For all $i \in \{0,1\}^n$, all $x \notin [0, i]$, we have that

  $$\mathsf{PRF}(K, x) = \mathsf{PRF}(K\{i+1\}, x)$$

  where $K \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $K\{i+1\} \leftarrow \mathsf{RangePuncture}(K, i)$. In other words, non-punctured keys and punctured key always agree on a point x, so long as $x \geq i$, where $[0, i]$ is the range where the key was punctured.

- (Iterative Puncturing) For $K \leftarrow \mathsf{KeyGen}(1^\lambda)$ and $K\{i\} \leftarrow \mathsf{RangePuncture}(K, i-1)$, it holds that

  $$\mathsf{RangePuncture}(K, i) = \mathsf{IterPuncture}(K\{i\}, i).$$

- (Compactness) The size of *any* punctured key $K\{i\}$ is $\mathsf{poly}(\lambda) \cdot O(n)$.

Finally, we state the security that we require for range puncturable PRFs, which is identical to the standard definition from [SW14].

Definition 5.2 (Pseudorandomness at Punctured Points)**.** There exists a negligible function $\mu(\cdot)$ such that for every PPT adversaries $\mathcal{A}$ and for every $i \in \{0,1\}^{n(\lambda)}$ and $x \leq i$ it holds that

$$|\Pr[\mathcal{A}(K\{i+1\}, \mathsf{PRF}(K,x))] - \Pr[\mathcal{A}(K\{i+1\}, u)]| \leq \mu(\lambda).$$

where $K \leftarrow \mathsf{KeyGen}(1^\lambda)$, $K\{i+1\} \leftarrow \mathsf{RangePuncture}(K,i)$, and $u \leftarrow \{0,1\}^{m(\lambda)}$.

We stress that, although range puncturable PRF are syntactically similar to standard puncturable PRFs, they are not directly implied by the construction in [SW14]. Even though we are allowed to puncture a PRF at polynomially many points, the size of punctured key will scale with the number of points that we have punctured, violating the compactness requirement. In the following we show how to modify the GGM tree method [GGM86] for building range puncturable PRFs.

5.1 Construction

We first recall the GGM tree construction of a normal PRF from a length-doubling PRG $G : \{0,1\}^\lambda \to \{0,1\}^{2\lambda}$. Given an output $G(s)$, we are going to divide it into two halves $G_0(s)$ and $G_1(s)$. First, we define the PRF for a single bit as:

$$\mathsf{PRF}(s,b) = G_b(s),\ b \in \{0,1\}.$$

Then we can recursively define the PRF as:

$$\mathsf{PRF}(s, x||b) = G_b(\mathsf{PRF}(s,x)),\ b \in \{0,1\}.$$

We can observe that the GGM construction of PRF forms a tree. For n-bit input, we will have n levels of the tree. The leaves of the tree correspond to the output of the PRF. Note that if the input has size λ, the tree is exponentially big. Therefore, we cannot compute the entire tree. However, we can still efficiently compute each particular input to the PRF.

Range Puncturable PRF. Recall that the range punctured key for a range puncturable PRF allows us to evaluate the PRF tree everywhere but a range $[0,i]$. Now we describe the construction of range puncturable PRF.

- $\mathsf{KeyGen}(1^\lambda)$: Output the PRG seed s as the PRF key $K = s$.
- $\mathsf{RangePuncture}(K,i)$: Let $i = i_0 i_1 \dots i_{n-1}$.

  If $i_0 = 0$, $\mathsf{PRF}(K,1)$ is included in $K\{i+1\}$. If $i_1 = 0$, $\mathsf{PRF}(K, i_0 1)$ is included in $K\{i+1\}$. Similarly, if $i_{n-1} = 0$, $\mathsf{PRF}(K, i_0 i_1 \dots i_{n-2} 1)$ is included in $K\{i+1\}$.

  $$K\{i+1\} = \{\mathsf{PRF}(s, i_0 i_1 \dots i_{k-1} 1) | i_k = 0,\ k \in [0, n-1],\ i = i_0 i_1 \dots i_{n-1}\}$$

  Output the range punctured key $K\{i+1\}$.

- $\mathsf{IterPuncture}(K\{i\}, i)$: Let $i - 1 = i_0 i_1 \dots i_{n-1}$.

  If $i_{n-1} = 0$, in order to puncture out range $[0, i]$ for $K\{i\}$, remove $\mathsf{PRF}(K, i_0 i_1 \dots i_{n-2} 1)$ from $K\{i\}$ and output the result as $K\{i+1\}$.

  If $i_{n-1} = 1$, assume there are ℓ consecutive ones in the least significant bits of i. Without loss of generality, we can further assume that $\ell < n$ (otherwise $i - 1 = 2^{n-1}$ and therefore $\mathsf{PRFPuncture}(i)$ is already empty). Therefore,

$$\begin{aligned} i - 1 &= i_0 i_1 \dots i_{n-\ell-1} i_{n-\ell} \dots i_{n-1} \\ &= i_0 i_1 \dots 0 \underbrace{1 \dots 1}_{\ell \text{ times}}. \end{aligned}$$

  Remove $\mathsf{PRF}(K, i_0 i_1 \dots i_{n-\ell-2} 1)$ from $K\{i\}$, append the following points

$$\begin{aligned} &\mathsf{PRF}(K, i_0 i_1 \dots i_{n-\ell-2} 1 i_{n-\ell}), \\ &\mathsf{PRF}(K, i_0 i_1 \dots i_{n-\ell-2} 1 i_{n-\ell} i_{n-\ell+1}), \\ &\dots, \\ &\mathsf{PRF}(K, i_0 i_1 \dots i_{n-\ell-2} 1 i_{n-\ell} i_{n-\ell+1} \dots i_{n-1}). \end{aligned}$$

  Which can all be computed starting from $\mathsf{PRF}(K, i_0 i_1 \dots i_{n-\ell-2} 1)$ from $K\{i\}$. Output the result as $K\{i+1\}$.

Note that the values we include in $K\{i+1\}$ are sufficient to compute the PRF for any value larger than i, since we can expand the tree below the neighboring nodes by applying the PRG to the values of the nodes. Furthermore, $K\{i+1\}$ can be computed efficiently and its size is $O(n)$ for any index i. We summarize the results of this section in the following theorem, appealing to the fact that PRGs can be constructed from any one-way function [HILL99].

Theorem 5.3. If one-way functions exist, then for all efficiently computable functions $n(\lambda)$ and $m(\lambda)$, there exists a range puncturable PRF family that maps $n(\lambda)$ bits to $m(\lambda)$ bits.

Proof. Since the security proof is very close to that of the GGM construction, we only provide a sketch here. We need to argue that given the range punctured key, an adversary cannot distinguish between the values at the punctured interval and truly random values. By PRG security, an adversary cannot distinguish between output of the PRG and a truly random value.

Let $i = i_0 i_1 \dots i_{n-1}$ and

$$K\{i+1\} = \{\mathsf{PRF}(s, i_0 i_1 \dots i_{k-1} 1) | i = i_0 i_1 \dots i_{n-1},\ i_k = 0,\ k \in [0, n-1]\}$$

be the range punctured key of interval $[0, i]$. Our goal is to replace the value $\mathsf{PRF}(s, x)$ for a given $x = x_0 x_1 \dots x_{n-1} \leq i$ with a uniformly random value. First, we can replace $\mathsf{PRF}(s, 0) = G_0(s)$, $\mathsf{PRF}(s, 1) = G_1(s)$ with s_0, s_1 uniformly sampled from $\{0,1\}^\lambda$ by the security of PRG. If $i_0 = 0$, we can then output s_1 as part of the $K\{i+1\}$ and use s_0 to compute the value of x. If $i_0 = 1$ and $x_0 = 0$, we can use s_0 to compute the value of x. Otherwise if $i_0 = 1$ and $x_0 = 1$, we need to compute the value of x with s_1. With s_b, for $b \in \{0,1\}$, as a truly random value, we can apply PRG security again to replace $\mathsf{PRF}(s, b0) = G_0(s_b)$, $\mathsf{PRF}(s, b1) = G_1(s_b)$ with random values s_{b0} and s_{b1}. Repeat this process until we arrive at the leaf of the tree corresponding to x. Indistinguishability follows by a standard hybrid argument. Now we have replaced the PRG values of the path from the root to x and its neighbour with uniformly random ones that are independent of each other. Since the value of x is now independent of the rest of the tree, $K\{i+1\}$ does not give the adversary any information about x. □

6 Sublinear Randomized Encodings

In the following, we present a construction of randomized encoding for the T-folded repeated application of a function f, where the complexity of the encoding algorithm, and consequently the size of the encoding, is $O(\sqrt{T})$.

Ingredients. For a given function $f : \mathcal{X} \to \mathcal{X}$, we assume the existence of the following cryptographic building blocks:

- A depth-independent reusable garbling scheme ($\mathsf{rGC.Garble}, \mathsf{rGC.Enc}, \mathsf{rGC.Eval}$).
- A fully-homomorphic encryption scheme $\mathsf{FHE} = (\mathsf{FHE.KeyGen}, \mathsf{FHE.Enc}, \mathsf{FHE.Eval}, \mathsf{FHE.Dec})$ with linear decrypt-and-multiply with noise bound B and prime modulus q.
- A pseudorandom generator $\mathsf{PRG} : \{0,1\}^\lambda \to \mathbb{Z}_q^{n \times nk\lceil \log q \rceil} \times \mathbb{Z}_q^n$.
- A range puncturable pseudorandom function ($\mathsf{PRF.KeyGEn}, \mathsf{PRF.RangePuncture}, \mathsf{PRF.IterPuncture}$) such that
$$\mathsf{PRF} : \mathcal{K} \times \left[\sqrt{T}\right] \to \{0,1\}^{4\lambda}.$$

Parameters. For convenience, we list all the parameters that we use in our scheme, and we discuss the constraints introduced by the subsequent construction:

- The security parameter λ.
- The lattice dimension n governs the hardness of the underlying LWE problem, and we set it to be some fixed $\mathsf{poly}(\lambda)$.
- The noise bound B determines (a bound on) the magnitude of the noise of the LWE problem, and we will set it to be as large as possible, conditioned on satisfying the constraint below.
- The modulus q is chosen to be sufficiently large, for instance:
$$q/4 = 2^{\omega(\log \lambda)} \cdot B \cdot (n \cdot k \cdot \lceil \log(q) \rceil + 1). \tag{1}$$
- The output size k is set to be the size of the output of Γ, a circuit that we evaluate in our construction (see below for a precise definition). As we shall see shortly, we set it to be some fixed $\mathsf{poly}(\lambda)$.
- The number of repetitions T.

Given the above parameter settings, we will henceforth refer to the LWE problem as the instance of $\mathrm{LWE}_{n,k,q,\chi}$ with parameters set to satisfy the above constraints (and in particular Equation 1), but otherwise we are free to choose parameters so that the problem is hard.

Construction. In our construction, we define for convenience the most significant bit function $\mathsf{MSB}(\cdot)$ as the operator that takes as input a vector in $\mathbb{Z}_q^k$ and return the most significant bit component-wise. On input the security parameter 1^λ, a function f, and an input $x \in \mathcal{X}$, the *encoding* algorithm proceeds as follows.

- Sample a matrix $\mathbf{A} \leftarrow \mathbb{Z}_q^{k \times n}$ and a key $K \leftarrow \mathcal{K}$ for a range puncturable PRF.
- Compute $(\tilde{C}, \mathsf{pk}) \leftarrow \mathsf{rGC.Garble}\left(1^\lambda, C\right)$ where the circuit C is defined in Figure 1.
- For $i = 1, \ldots, \sqrt{T} + 1$:
  - Let $(\mathsf{o}_i, \mathsf{r}_i, \mathsf{s}_i, \mathsf{k}_i) \leftarrow \mathsf{PRF}(K, i)$.
  - Expand $(\mathbf{R}_i, \mathbf{t}_i) \leftarrow \mathsf{PRG}(\mathsf{o}_i)$.
  - Compute the key pair $(\mathsf{sk}_i, \mathsf{pk}_i) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda; \mathsf{k}_i)$ and parse $\mathsf{sk}_i = (\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,n})$.
  - Compute $\mathbf{z}_i = \mathsf{MSB}(\mathbf{A}\mathbf{t}_i)$ and
    $$\mathbf{B}_i = \mathbf{A}\mathbf{R}_i + \mathbf{E}_i + (\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,n}) \otimes \mathbf{G}$$
    where $\mathbf{E}_i \leftarrow [-B, B]^{k \times nk\lceil \log q \rceil}$ is a randomly sampled noise matrix and $\mathbf{G}$ is the gadget matrix, i.e., $\mathbf{G} = \mathbf{g} \otimes \mathbf{I}_k$, where $\mathbf{g} = (1, 2, 4, , \ldots, 2^{\lceil \log q \rceil - 1})$.
  - Compute $e_{i,part} \leftarrow \mathsf{rGC.PartEnc}(\mathsf{pk}, \mathsf{r}_i)$.
- The garbled circuit is defined as
  $$\mathbf{A}, \tilde{C}, \mathsf{pk}, \{\mathsf{pk}_i, \mathbf{B}_i, \mathbf{z}_i, e_{i,part}\}_{i \in [\sqrt{T}]}$$
  whereas the garbled input is defined as
  $$c_1 \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}_1, (x, K\{2\}); \mathsf{s}_1) \quad \text{and} \quad e_1 \leftarrow \mathsf{rGC.Enc}(\mathsf{pk}, (\mathsf{o}_1, \mathsf{pk}_1, x, 1, c_1); \mathsf{r}_1).$$

Next we describe the behavior of the *decoding* algorithm. On input a garbled circuit and a garbled input as specified above, the algorithm proceeds as follows. For $i = 1, \ldots, \sqrt{T}$:

- Compute
  $$c \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}_i, \Gamma_i, c_i)$$
  and let $\mathbf{L}_c$ be the vector concatenation corresponding to the coefficients of the linear function $\mathsf{Dec\&Mult}(\cdot, c, \lceil q/2 \rceil)$.
- Decode $\mathbf{h}_i \leftarrow \mathsf{rGC.Eval}(\tilde{C}, C, e_i)$.
- Set
  $$(c_{i+1}, e_{i+1,input}) = \mathsf{MSB}\left(\mathbf{B}_i\mathbf{G}^{-1}(\mathbf{L}_c) - \mathbf{A}\mathbf{h}_i\right) \oplus \mathbf{z}_i$$
  and $e_{i+1} = \mathsf{rGC.RecEnc}(e_{i+1,part}, e_{i+1,input})$.

The output of the decoding algorithm is defined to be $\mathsf{rGC.Eval}\left(\tilde{C}, C, e_{\sqrt{T}+1}\right)$.

Circuit $C(\mathsf{o}_i, \mathsf{pk}_i, x, i, c_i)$

- If $i = \sqrt{T} + 1$: Return x.
- Otherwise, expand $(\mathbf{R}_i, \mathbf{t}_i) \leftarrow \mathsf{PRG}(\mathsf{o}_i)$.
- Compute the ciphertext

$$c \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}_i, \Gamma_i, c_i)$$

  and let $\mathbf{L}_c$ be the vector concatenation corresponding to the coefficients of the linear function $\mathsf{Dec\&Mult}(\cdot, c, \lceil q/2 \rceil)$. Here, the circuit $\Gamma_i(x, K\{i+1\})$ is defined as follows:
  - Compute $y = f_{\sqrt{T}}(x)$.
  - Expand $(\mathsf{o}_{i+1}, \mathsf{r}_{i+1}, \mathsf{s}_{i+1}, \mathsf{k}_{i+1}) \leftarrow \mathsf{PRF}(K\{i+1\}, i+1)$.
  - Recompute $(\mathsf{sk}_{i+1}, \mathsf{pk}_{i+1}) \leftarrow \mathsf{FHE.KeyGen}(1^\lambda; \mathsf{k_{i+1}})$.
  - Puncture the key $K\{i+2\} \leftarrow \mathsf{PRF.IterPuncture}(K\{i+1\}, i+1)$.
  - Return a ciphertext

$$c_{i+1} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}_{i+1}, (y, K\{i+2\}); \mathsf{s}_{i+1})$$

    and an encoding

$$e_{i+1,input} \leftarrow \mathsf{rGC.InputEnc}((\mathsf{o}_{i+1}, \mathsf{pk}_{i+1}, y, i+1, c_{i+1}), \mathsf{r}_{i+1}).$$

- Return

$$\mathbf{h}_i = \mathbf{R}_i \mathbf{G}^{-1}(\mathbf{L}_c) - \mathbf{t}_i.$$

Figure 1: The circuit C.

Efficiency. In the following we argue that the scheme satisfies efficiency, and in particular that the runtime to the encoding algorithm is bounded by some polynomial $\mathsf{poly}(\lambda) \cdot \sqrt{T}$. By construction, we set k to be the size of the output of the circuit Γ and we argue that k can be bounded by some fixed polynomial in λ. Zooming in the components of the output of Γ, we see that they consist of:

- An FHE ciphertext c_{i+1} encrypting the updated input $y = f(x)$, which is of size exactly $\log(|\mathcal{X}|) = \mathsf{poly}(\lambda)$, and a punctured key which, by compactness of the range puncturable PRF, is also of size $\mathsf{poly}(\lambda)$. By the compactness of the FHE evaluation algorithm, the overall size of the ciphertext can be bounded by $|c_{i+1}| = \mathsf{poly}(\lambda)$.
- An input encoding $e_{i,input}$, whose size is at most the size of an encoding e_i of the reusable garbling scheme. The size of an encoding depends (ignoring the security parameter), on the size of the input and on the size of the output of the circuit C. The input size of C can be bounded by $\mathsf{poly}(\lambda)$ since it consists only of five components whose individual size is also bounded by $\mathsf{poly}(\lambda)$. In more details:
  - $|\mathsf{o}_i| = \lambda$ by definition.
  - $|\mathsf{pk}_i| = \mathsf{poly}(\lambda)$ by the compactness of the FHE scheme.
  - $|x| = \log(|\mathcal{X}|) = \mathsf{poly}(\lambda)$.
  - $|i| \leq \log(T) \leq \lambda$.
  - $|c_i| = \mathsf{poly}(\lambda)$ as discussed above.

  The size of the output of C is $\log(q) \cdot n = \mathsf{poly}(\lambda)$.

Overall, we have established that $k = \mathsf{poly}(\lambda)$. Therefore, all components of the output of the encoding algorithm are also bounded by some fixed polynomial in the security parameter. Since there are exactly $4\sqrt{T} + 5$ such components, the bound on the output size follows. Similarly, we can easily see that the runtime of the encoding algorithm is dominated by:

- The $\sqrt{T}+1$ iterations needed to compute $\{pk_i, \mathbf{B}_i, \mathbf{z}_i, e_{i,part}\}_{i \in [\sqrt{T}]}$, where the runtime of each iteration depends polynomially on λ (and nothing else).
- The invocation of $\mathsf{rGC.Garble}$ on input C, whose runtime grows with $\mathsf{poly}(\lambda) \cdot |C|$. The size of C can be bounded by $\mathsf{poly}(\lambda) \cdot \sqrt{T}$, since the only computation that depends on T is the evaluation of $f_{\sqrt{T}}(x)$.

Overall, we can bound the runtime of the encoding algorithm by $\mathsf{poly}(\lambda)\sqrt{T}$, and the efficiency claim follows.

Correctness. Let us define $y_i = f_{\sqrt{T}}(y_{i-1})$, where $y_1 = x$. For $i = 1, \ldots, \sqrt{T} + 1$, we prove by induction that c_i, e_i are of the following form:

$$c_i = \mathsf{FHE.Enc}(\mathsf{pk}_i, (y_i, K\{i+1\}); \mathsf{s}_i),$$
$$e_i = \mathsf{rGC.Enc}(\mathsf{pk}, (\mathsf{o}_i, \mathsf{pk}_i, y_i, i, c_i); \mathsf{r}_i).$$

For $i = 1$, this holds by the definition of garbled input. Assume this holds for i, we show next that c_{i+1}, e_{i+1} are the same form. Recall that

$$\begin{aligned}\mathbf{h}_i &= \mathsf{rGC.Eval}(\tilde{C}, C, e_i) \\ &= \mathbf{R}_i \mathbf{G}^{-1}(\mathbf{L}_c) - \mathbf{t}_i\end{aligned}$$

where $c = \mathsf{FHE.Eval}(\mathsf{pk}_i, \Gamma_i, c_i)$. Substituting, we obtain:

$$\begin{aligned}
\mathsf{MSB}(\mathbf{B}_i\mathbf{G}^{-1}(\mathbf{L}_c) - \mathbf{A}\mathbf{h}_i) &= \mathsf{MSB}((\mathbf{A}\mathbf{R}_i + \mathbf{E}_i + (\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,n}) \otimes \mathbf{G})\mathbf{G}^{-1}(\mathbf{L}_c) - \mathbf{A}\mathbf{h}_i) \\
&= \mathsf{MSB}(\mathbf{A}\mathbf{R}_i\mathbf{G}^{-1}(\mathbf{L}_c) + \mathbf{E}_i\mathbf{G}^{-1}(\mathbf{L}_c) + \mathbf{L}_c(\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,n}) - \mathbf{A}\mathbf{h}_i) \\
&= \mathsf{MSB}(\mathbf{A}\mathbf{R}_i\mathbf{G}^{-1}(\mathbf{L}_c) + \mathbf{e} + \mathbf{L}_c(\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,n}) - \mathbf{A}(\mathbf{R}_i\mathbf{G}^{-1}(\mathbf{L}_c) - \mathbf{t}_i)) \\
&= \mathsf{MSB}(\mathbf{L}_c(\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,n}) + \mathbf{A}\mathbf{t}_i + \mathbf{e}) \\
&= \mathsf{MSB}(\mathrm{Dec\&Mult}((\mathbf{s}_{i,1}, \ldots, \mathbf{s}_{i,n}), c, \lceil q/2 \rceil) + \mathbf{A}\mathbf{t}_i + \mathbf{e}).
\end{aligned}$$

Therefore

$$\begin{aligned}
\mathsf{MSB}\left(\mathbf{B}_i\mathbf{G}^{-1}(\mathbf{L}_c) - \mathbf{A}\mathbf{h}_i\right) \oplus \mathbf{z}_i &= \mathsf{MSB}(\lceil q/2 \rceil \cdot \Gamma_i(y_i, K\{i+1\}) + \tilde{\mathbf{e}} + \mathbf{A}\mathbf{t}_i + \mathbf{e}) \oplus \mathbf{z}_i \\
&= (c_{i+1}, e_{i+1,input}) \oplus \mathsf{MSB}(\mathbf{A}\mathbf{t}_i + \mathbf{e} + \tilde{\mathbf{e}}) \oplus \mathbf{z}_i \\
&= (c_{i+1}, e_{i+1,input}) \oplus \mathsf{MSB}(\mathbf{A}\mathbf{t}_i) \oplus \mathsf{MSB}(\mathbf{A}\mathbf{t}_i) \\
&= (c_{i+1}, e_{i+1,input}).
\end{aligned}$$

To show the above equality, we claim that $\mathsf{MSB}(\mathbf{A}\mathbf{t}_i + \mathbf{e} + \tilde{\mathbf{e}}) = \mathsf{MSB}(\mathbf{A}\mathbf{t}_i)$, with all but negligible probability. Observe that

$$||\mathbf{e} + \tilde{\mathbf{e}}||_\infty \leq \tilde{B} := B \cdot (n \cdot k \cdot \lceil \log(q) \rceil + 1).$$

For a given $v \in \mathbb{Z}_q$ say that v is bad if $|v - q/4| < \tilde{B}$ or $|v + q/4| < \tilde{B}$. By choosing q sufficiently large, e.g., by setting $q/4 > 2^{\omega(\log \lambda)} \cdot \tilde{B}$, we get that the probability that a uniformly random $v \in \mathbb{Z}_q$ is bad is negligible. We further define that a vector $\mathbf{v} \in \mathbb{Z}_q^k$ is bad if any of its components is bad. Fix any $\mathbf{v} \neq 0$, and let $\mathbf{a}_i, \ldots, \mathbf{a}_k$ be the rows of $\mathbf{A}$. Since $\mathbf{a}_i$ is uniformly random, the probability that $\mathbf{a}_i^\intercal \mathbf{v}$ is bad is negligible. Therefore, by a union bound, the probability that $\mathbf{A}\mathbf{v}$ is bad is negligible as well. We can conclude that $\mathsf{MSB}(\mathbf{A}\mathbf{t}_i + \mathbf{e} + \tilde{\mathbf{e}}) = \mathsf{MSB}(\mathbf{A}\mathbf{t}_i)$ with overwhelming probability. Finally, we can see that that

$$e_{i+1} = \mathsf{rGC.RecEnc}(e_{i+1,part}, e_{i+1,input}) = \mathsf{rGC.Enc}(\mathsf{pk}, (\mathsf{o}_{i+1}, \mathsf{pk}_{i+1}, y_{i+1}, i+1, c_{i+1}); \mathsf{r}_{i+1}).$$

proving the above claim. Taking $i = \sqrt{T} + 1$, we have that

$$e_{\sqrt{T}+1} = \mathsf{rGC.Enc}(\bar{\mathsf{pk}}, (\mathsf{o}_{\sqrt{T}+1}, pk_{\sqrt{T}+1}, y_{\sqrt{T}+1}, \sqrt{T}+1, c_{\sqrt{T}+1}); \mathsf{r}_{\sqrt{T}+1})$$

and by the correctness of the reusable garbled circuit we obtain that

$$\begin{aligned}
\mathsf{rGC.Eval}\left(\tilde{C}, C, e_{\sqrt{T}+1}\right) &= C(\mathsf{o}_{\sqrt{T}+1}, pk_{\sqrt{T}+1}, y_{\sqrt{T}+1}, \sqrt{T}+1, c_{\sqrt{T}+1}) \\
&= y_{\sqrt{T}+1} \\
&= \underbrace{f_{\sqrt{T}}(\ldots f_{\sqrt{T}}(f_{\sqrt{T}}}_{\sqrt{T} \text{ times}}(x)) \ldots) \\
&= f_T(x).
\end{aligned}$$

Security Analysis. In the following we present our main theorem.

Theorem 6.1. Let rGC be a reusable garbling scheme, FHE be a semantically secure fully-homomorphic encryption scheme, PRG be a pseudorandom generator, and PRF be a range puncturable pseudorandom function. Then the construction described above is a secure randomized encoding.

Proof. Fix any two x_0 and x_1 such that $y = f_T(x_0) = f_T(x_1)$. We can show that their randomized encodings are computationally indistinguishable by defining a series of hybrid experiments. We define the very first experiment to be the original distribution, except with the input fixed to x_0. For $i = 1, \ldots, \sqrt{T}+1$ we define a sequence of seven sub-hybrids, where we prove indistinguishability inductively.

- Hybrid $(i, 1)$: We substitute the point $(\mathsf{o}_i, \mathsf{r}_i, \mathsf{s}_i, \mathsf{k}_i) \leftarrow \mathsf{PRF}(K, i)$ with a uniformly random one. By induction hypothesis, at this hybrid, the only information that the adversary has about the PRF key is the punctured key

$$K\{i+1\} \leftarrow \mathsf{PRF.IterPuncture}(K\{i\}, i),$$

  where $K\{1\} = K$. Since $i < i + 1$, by the pseudorandomness of the range puncturable PRF, even given $K\{i + 1\}$, $\mathsf{PRF}(K, i)$ is still computationally indistinguishable from a uniformly random one. It follows that this hybrid experiement is computationally indistinguishable from the previous one.

- Hybrid $(i, 2)$: We replace $e_{i,part}$ and $e_{i,input}$ with the output of $\mathsf{rGC.SimEnc}(e_i)$ where $e_i = \mathsf{rGC.Enc}(\mathsf{pk}, (\mathsf{o}_i, \mathsf{pk}_i, y_i, i, c_i); \mathsf{r}_i)$. By the simulatability of partial encodings, this hybrid is identical to the previous one.

- Hybrid $(i, 3)$: We replace e_i by $\mathsf{rGC.Sim}(1^\lambda, C, \mathsf{pk}, \mathbf{h}_i)$. By the simulatability of reusable garbled circuits, the distribution induced by this hybrid is computationally indistinguishable from the previous one.

- Hybrid $(i, 4)$: We switch

$$(\mathbf{R}_i, \mathbf{t}_i) \leftarrow \mathsf{PRG}(\mathsf{o}_i)$$

  with uniformly random elements. Since o_i is uniformly random, the security of the PRG guarantees that $(\mathbf{R}_i, \mathbf{t}_i)$, expanded from o_i, are computationally indistinguishable from uniformly random elements. Therefore, this hybrid is computationally indistinguishable from the previous one.

- Hybrid $(i, 5)$: In this hybrid the hint $\mathbf{h}_i$ is sampled uniformly from $\mathbb{Z}_q^n$, and $\mathbf{z}_i$ is hardwired with the output $(c_{i+1}, e_{i+1,input})$:

$$\mathbf{z}_i = \mathsf{MSB}\left(\mathbf{B}_i\mathbf{G}^{-1}(\mathbf{L}_c) - \mathbf{A}\mathbf{h}_i\right) \oplus (c_{i+1}, e_{i+1,input}).$$

  This hybrid is identical to the previous one, since $(\mathbf{h}_i, \mathbf{z}_i)$ has the same conditional distribution.

- Hybrid $(i, 6)$: We choose a uniformly random $\mathbf{B}_i$ and keep everything else the same. By the LWE assumption, this modification is computationally undetecatable .

- Hybrid $(i, 7)$: We switch

$$c_i \leftarrow \mathsf{FHE.Enc}(pk_i, (y, K\{i+1\}); \mathsf{s}_i)$$

with an encryption of 0:

$$c_i \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}_i, 0; \mathsf{s}_i).$$

By the semantic security of the FHE scheme, it follows that this hybrid is computationally indistinguishable from the previous one. We note that in this hybrid we have removed $K\{i+1\}$ from the view of the attacker and hence maintained the invariant that during hybrid series i, the distinguisher has only access to $K\{i+1\}$ but not prior keys.

After proceeding with this series of hybrids, at the very last one, we have just hardwired the output of the computation y and have remove all keys from the view of the adversary. Because $y = f_T(x_0) = f_T(x_1)$, we can then proceed with the same series of hybrid backwards, but using x_1 as input instead. This shows that the two distributions are computationally indistinguishable and concludes our proof. □

7 Garbled RAM with Sublinear Encodings

Before we describe sublinear garbled RAM, let us fix some notation for describing standard RAM computation. We use the notation $P^D(x)$ to denote the execution of program P that takes a short input x and has random-access to a memory of size n, which may initially contain some data $D \in \{0,1\}^n$. The program can read/write to various locations in memory throughout the execution. A useful representation of a RAM program P is through Step Circuit which executes a single step of RAM computation:

$$C^P(\text{state}, b^{\text{read}}) = (\text{state}', i_{\text{read}}, i_{\text{write}}, b_{\text{write}}).$$

This circuit takes as input the current state and a bit b^{read} residing in the the last read memory location. It outputs an updated state$'$, the next location to read $i_{\text{read}} \in [n]$, a location to write to $i_{\text{write}} \in [n] \cup \{\perp\}$ (where $\perp$ values are ignored), a bit b^{write} to write into that location.

The computation $P^D(x)$ starts in the initial state $\text{state}_1 = x$ corresponding to the input and all other values set to 0, by convention. In each step j, the computation proceeds by running $C^P(\text{state}_j, b_j^{\text{read}}) = (\text{state}_{j+1}, i_{\text{read}}, i_{\text{write}}, b_{\text{write}})$. The program then reads the requested location i_{read} by setting $b_{j+1}^{\text{read}} = D[i_{\text{read}}]$ and, if $i_{\text{write}} \neq \perp$, we overwrite the location by setting $D[i_{\text{write}}] = b_{\text{write}}$. The value $y =$ state output by the last step serves as the output of the computation, and we assume here that $|x| = |y|$ for convenience.

Sublinear Garbled RAM. In this work, we are interested in constructing *one-time* sublinear garbled RAM programs, where the runtime of the client is sublinear in the number of steps (T) but can otherwise grow with the size of the memory of the computation (and in particular $|D|$), which is often referred to as the settings *without persistent memory*. We omit formal definitions of garbled RAM, and we refer the reader to [GHRW14] for a more comprehensive treatment of this notion. For us, it suffices to observe that we can parse the computation of a RAM program as the T-folded execution of a circuit Γ with the syntax

$$\Gamma(D, \text{state}, i_{\text{read}}, i_{\text{write}}, b_{\text{read}}, b_{\text{write}}) = (D', \text{state}', i'_{\text{read}}, i'_{\text{write}}, b'_{\text{read}}, b'_{\text{write}})$$

which is defined as the combination of C^P and P^D, as described above, and T is a bound on the worst-case number of steps of the computation. To garble RAM computation using our sublinear randomized encoding (Section 6) it suffices to invoke the Enc algorithm on input the circuit Γ and the input $(0^{|D|}, x, 0, 0, 0, 0)$. Security follows directly from the simulatability of the randomized encoding.

Efficiency. The running time of the client is identical to that of the encoding procedure, and therefore can be upper bounded by $\mathsf{poly}(\lambda, |x|, |D|) \cdot \sqrt{T}$. On the other hand, the runtime of the server is identical to that of the decoding algorithm, and therefore bounded by $\mathsf{poly}(\lambda, |x|, |D|) \cdot T$. Therefore, we obtain the first garbled RAM scheme with client complexity sublinear in T, without using obfuscation.

Acknowledgements

G.M. wishes to thank Chris Peikert for inspiring discussions on timed cryptography at an early stage of this work.

S.A. was supported by the CyStar center of excellence and a DST Swarnajayanti fellowship. G.M. was supported by the European Research Council through an ERC Starting Grant (Grant agreement No. 101077455, ObfusQation). T.Z. was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy - EXC 2092 CASA – 390781972.

References

[ACL+22] Martin R Albrecht, Valerio Cini, Russell WF Lai, Giulio Malavolta, and Sri AravindaKrishnan Thyagarajan. Lattice-based snarks: Publicly verifiable, preprocessing, and recursively composable. In *Annual International Cryptology Conference*, pages 102–132. Springer, 2022.

[Agr19] Shweta Agrawal. Indistinguishability obfuscation without multilinear maps: new methods for bootstrapping and instantiation. In *Advances in Cryptology–EUROCRYPT 2019: 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19–23, 2019, Proceedings, Part I 38*, pages 191–225. Springer, 2019.

[AJ15] Prabhanjan Ananth and Abhishek Jain. Indistinguishability obfuscation from compact functional encryption. In *Annual Cryptology Conference*, pages 308–326. Springer, 2015.

[AK21] Aydin Abadi and Aggelos Kiayias. Multi-instance publicly verifiable time-lock puzzle and its applications. In *Financial Cryptography and Data Security: 25th International Conference, FC 2021, Virtual Event, March 1–5, 2021, Revised Selected Papers, Part II 25*, pages 541–559. Springer, 2021.

[ASP13] Jacob Alperin-Sheriff and Chris Peikert. Practical bootstrapping in quasilinear time. In Ran Canetti and Juan A. Garay, editors, *Advances in Cryptology – CRYPTO 2013*, pages 1–20, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[BDF21] Jeffrey Burdges and Luca De Feo. Delay encryption. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 302–326. Springer, 2021.

[BDGM19] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Leveraging linear decryption: Rate-1 fully-homomorphic encryption and time-lock puzzles. In *Theory of Cryptography Conference*, pages 407–437. Springer, 2019.

[BDGM23] Zvika Brakerski, Nico Döttling, Sanjam Garg, and Giulio Malavolta. Candidate io from homomorphic encryption schemes. *Journal of Cryptology*, 36(3):27, 2023.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 345–356, 2016.

[BGL+15] Nir Bitansky, Sanjam Garg, Huijia Lin, Rafael Pass, and Sidharth Telang. Succinct randomized encodings and their applications. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 439–448, 2015.

[BN00] Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *Advances in Cryptology — CRYPTO 2000*, pages 236–254, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

[BV14] Zvika Brakerski and Vinod Vaikuntanathan. Lattice-based fhe as secure as pke. In *Proceedings of the 5th Conference on Innovations in Theoretical Computer Science*, ITCS '14, page 1–12, New York, NY, USA, 2014. Association for Computing Machinery.

[BV18] Nir Bitansky and Vinod Vaikuntanathan. Indistinguishability obfuscation from functional encryption. *Journal of the ACM (JACM)*, 65(6):1–37, 2018.

[CCC+15] Yu-Chi Chen, Sherman SM Chow, Kai-Min Chung, Russell WF Lai, Wei-Kai Lin, and Hong-Sheng Zhou. Computation-trace indistinguishability obfuscation and its applications. *IACR Cryptol. ePrint Arch.*, 2015:406, 2015.

[CH16] Ran Canetti and Justin Holmgren. Fully succinct garbled ram. In *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science*, pages 169–178, 2016.

[CHJV15] Ran Canetti, Justin Holmgren, Abhishek Jain, and Vinod Vaikuntanathan. Succinct garbling and indistinguishability obfuscation for ram programs. In *Proceedings of the forty-seventh annual ACM symposium on Theory of Computing*, pages 429–437, 2015.

[CJJ22] Arka Rai Choudhuri, Abhihsek Jain, and Zhengzhong Jin. Snargs for p from lwe. In *2021 IEEE 62nd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 68–79. IEEE, 2022.

[CLM23] Valerio Cini, Russell WF Lai, and Giulio Malavolta. Lattice-based succinct arguments from vanishing polynomials. In *Annual International Cryptology Conference*, pages 72–105. Springer, 2023.

[DGM23] Jesko Dujmovic, Rachit Garg, and Giulio Malavolta. Time-lock puzzles with efficient batch solving. *Cryptology ePrint Archive*, 2023.

[DHMW23] Nico Döttling, Lucjan Hanzlik, Bernardo Magri, and Stella Wohnig. Mcfly: verifiable encryption to the future made practical. In *International Conference on Financial Cryptography and Data Security*, pages 252–269. Springer, 2023.

[FKPS21] Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part III 19*, pages 447–479. Springer, 2021.

[Gen09] Craig Gentry. *A fully homomorphic encryption scheme*. PhD thesis, Stanford, CA, USA, 2009. AAI3382729.

[GGHW17] Sanjam Garg, Craig Gentry, Shai Halevi, and Daniel Wichs. On the implausibility of differing-inputs obfuscation and extractable witness encryption with auxiliary input. *Algorithmica*, 79:1353–1373, 2017.

[GGM86] Oded Goldreich, Shafi Goldwasser, and Silvio Micali. How to construct random functions. *J. ACM*, 33(4):792–807, aug 1986.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 467–476, 2013.

[GHRW14] C. Gentry, S. Halevi, M. Raykova, and D. Wichs. Outsourcing private ram computation. In *2014 IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 404–413, Los Alamitos, CA, USA, oct 2014. IEEE Computer Society.

[GKP+13] Shafi Goldwasser, Yael Kalai, Raluca Ada Popa, Vinod Vaikuntanathan, and Nickolai Zeldovich. Reusable garbled circuits and succinct functional encryption. In *Proceedings of the 45th annual ACM symposium on Symposium on theory of computing*, STOC '13, pages 555–564, New York, NY, USA, 2013. ACM.

[GL89] Oded Goldreich and Leonid Levin. A hard-core predicate for all one-way functions. pages 25–32, 01 1989.

[GP21] Romain Gay and Rafael Pass. Indistinguishability obfuscation from circular security. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 736–749, 2021.

[GSW13] Craig Gentry, Amit Sahai, and Brent Waters. Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 75–92. Springer, 2013.

[GSZB23] Noemi Glaeser, István András Seres, Michael Zhu, and Joseph Bonneau. Cicada: A framework for private non-interactive on-chain auctions and voting. *Cryptology ePrint Archive*, 2023.

[HILL99] Johan HÅstad, Russell Impagliazzo, Leonid A. Levin, and Michael Luby. A pseudorandom generator from any one-way function. *SIAM Journal on Computing*, 28(4):1364–1396, 1999.

[HLL23] Yao-Ching Hsieh, Huijia Lin, and Ji Luo. Attribute-based encryption for circuits of unbounded depth from lattices. In *2023 IEEE 64th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 415–434. IEEE, 2023.

[JMRR21] Samuel Jaques, Hart Montgomery, Razvan Rosie, and Arnab Roy. Time-release cryptography from minimal circuit assumptions. In Avishek Adhikari, Ralf Küsters, and Bart Preneel, editors, *Progress in Cryptology – INDOCRYPT 2021*, pages 584–606, Cham, 2021. Springer International Publishing.

[KLX20] Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part III 18*, pages 390–413. Springer, 2020.

[LJKW18] Jia Liu, Tibor Jager, Saqib A Kakvi, and Bogdan Warinschi. How to build time-lock encryption. *Designs, Codes and Cryptography*, 86:2549–2586, 2018.

[LM23] Russell W. F. Lai and Giulio Malavolta. Lattice-based timed cryptography. In Helena Handschuh and Anna Lysyanskaya, editors, *Advances in Cryptology – CRYPTO 2023*, pages 782–804, Cham, 2023. Springer Nature Switzerland.

[LPS17] Huijia Lin, Rafael Pass, and Pratik Soni. Two-round concurrent non-malleable commitment from time-lock puzzles. *IACR Cryptol. ePrint Arch.*, 2017:273, 2017.

[MT19] Giulio Malavolta and Sri Aravinda Krishnan Thyagarajan. Homomorphic time-lock puzzles and applications. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology – CRYPTO 2019*, pages 620–649, Cham, 2019. Springer International Publishing.

[Pei09] Chris Peikert. Public-key cryptosystems from the worst-case shortest vector problem: Extended abstract. STOC '09, page 333–342, New York, NY, USA, 2009. Association for Computing Machinery.

[QWW18] Willy Quach, Hoeteck Wee, and Daniel Wichs. Laconic function evaluation and applications. In *2018 IEEE 59th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 859–870. IEEE, 2018.

[Reg09] Oded Regev. On lattices, learning with errors, random linear codes, and cryptography. *Journal of the ACM (JACM)*, 56(6):1–40, 2009.

[RSW96] Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto. 1996.

[SLM+23] Shravan Srinivasan, Julian Loss, Giulio Malavolta, Kartik Nayak, Charalampos Papamanthou, and Sri AravindaKrishnan Thyagarajan. Transparent batchable time-lock puzzles and applications to byzantine consensus. In Alexandra Boldyreva and

Vladimir Kolesnikov, editors, *Public-Key Cryptography – PKC 2023*, pages 554–584, Cham, 2023. Springer Nature Switzerland.

[SW14] Amit Sahai and Brent Waters. How to use indistinguishability obfuscation: Deniable encryption, and more. In *Proceedings of the Forty-Sixth Annual ACM Symposium on Theory of Computing*, STOC '14, page 475–484, New York, NY, USA, 2014. Association for Computing Machinery.

[TAF+23] Nirvan Tyagi, Arasu Arun, Cody Freitag, Riad Wahby, Joseph Bonneau, and David Mazières. Riggs: Decentralized sealed-bid auctions. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 1227–1241, 2023.

[TBM+20] Sri Aravinda Krishnan Thyagarajan, Adithya Bhat, Giulio Malavolta, Nico Döttling, Aniket Kate, and Dominique Schröder. Verifiable timed signatures made practical. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1733–1750, 2020.

[TCLM21] Sri Aravinda Krishnan Thyagarajan, Guilhem Castagnos, Fabian Laguillaumie, and Giulio Malavolta. Efficient cca timed commitments in class groups. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2663–2684, 2021.

[TMMS22] Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, and Pedro Moreno-Sanchez. Universal atomic swaps: Secure exchange of coins across all blockchains. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 1299–1316. IEEE, 2022.

[TMSS22] Sri AravindaKrishnan Thyagarajan, Giulio Malavolta, Fritz Schmid, and Dominique Schröder. Verifiable timed linkable ring signatures for scalable payments for monero. In *European Symposium on Research in Computer Security*, pages 467–486. Springer, 2022.

[Tsa22] Rotem Tsabary. Candidate witness encryption from lattice techniques. In Yevgeniy Dodis and Thomas Shrimpton, editors, *Advances in Cryptology – CRYPTO 2022*, pages 535–559, Cham, 2022. Springer Nature Switzerland.

[VWW22] Vinod Vaikuntanathan, Hoeteck Wee, and Daniel Wichs. Witness encryption and null-io from evasive lwe. In Shweta Agrawal and Dongdai Lin, editors, *Advances in Cryptology – ASIACRYPT 2022*, pages 195–221, Cham, 2022. Springer Nature Switzerland.

[WW21] Hoeteck Wee and Daniel Wichs. Candidate obfuscation via oblivious lwe sampling. In Anne Canteaut and François-Xavier Standaert, editors, *Advances in Cryptology – EUROCRYPT 2021*, pages 127–156, Cham, 2021. Springer International Publishing.

[WXDS20] Jun Wan, Hanshen Xiao, Srinivas Devadas, and Elaine Shi. Round-efficient byzantine broadcast under strongly adaptive and majority corruptions. In *Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18*, pages 412–456. Springer, 2020.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd Annual Symposium on Foundations of Computer Science, Chicago, Illinois, USA, 3-5 November 1982*, pages 160–164. IEEE Computer Society, 1982.

[Yao86] Andrew Chi-Chih Yao. How to generate and exchange secrets (extended abstract). In *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*, pages 162–167. IEEE Computer Society, 1986.